MONTE CARLO DETERMINATION OF BOUNDS ON ERROR CORRECTING CODES

RICHARD HAROLD HOLT

MONTE CARLO DETERMINATION OF BOUNDS

ON ERROR CORRECTING CODES

by

Richard Harold Holt
Lieutenant, United States Navy
B.S., Texas Western College
of the University of Texas, 1961

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING ELECTRONICS

from the

NAVAL POSTGRADUATE SCHOOL
June 1967

ABSTRACT

Analytical bounds on the capabilities of error correcting codes have been found for most known codes. There are only a limited number of coding theorists available for the solution of such problems; however, new coding techniques are constantly being proposed to meet new communication system problems. This paper develops and describes a Monte Carlo method for estimating bounds experimentally. A bound on the block error rate is developed for use in evaluating optimum codes. The random sampling technique evolved is used to evaluate the bounds on four representative error correcting codes. The close agreement between the theoretical and experimental results establishes confidence in the method's use to determine bounds and capabilities. The technique and problems described can also be used to simulate error correction in system coding studies.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I deeply appreciate the assistance of Professor C.F. Klamm, Jr., for his ideas, suggestions and advice about the use of Monte Carlo evaluation of error correcting codes.

CHAPTER 1

INTRODUCTION

During the past several years, there have been many remarkable developments and discoveries in the construction of error correcting codes. Many of these codes meet the essential requirement that they can be implemented with practical circuits. The bounds on most of these codes have been determined analytically, but the task is often quite complex. It is probable that there are coding schemes for which it would be extremely difficult to obtain bounds by analytical methods alone. In this paper, we shall endeavor to develop a method for estimating bounds experimentally.

Random sampling, or Monte Carlo, methods have proven valuable in providing solutions to many types of both probabilistic and deterministic problems. The Monte Carlo method will be used to verify the bounds on block error rates of four representative error correcting codes. The successful use of the method in cases with known results should establish our confidence in the method, and provide us with a stimulus for further use of the technique.

The purpose of this paper is not to advance coding theory or techniques; it is to provide a method for the experimental verification and/or determination of bounds on error correcting codes.

Some understanding of communication channels and probability theory is assumed on the part of the reader. Much time and effort has been devoted to the task of reducing the material to a direct and, hopefully, readily understandable presentation. The mathematics has been held to the minimum thought necessary for comprehension of the methods involved. This does not mean, however, that the algebra, finite field computations, probability and statistics involved in the understanding of error

correcting coding theory is simple. Considerable study was required to develop the background upon which this paper is based.

Chapter 2 discusses random number generators and the Monte Carlo method, and their use in this particular application.

Coding capabilities and bounds are briefly considered and discussed in Chapter 3. The effectiveness of error correction is sometimes measured by the comparison of bit error rates before and after correction; however, it was felt that for our purposes, initial bit error rate versus final block error rate would be a better measure of a code's effectiveness. In light of this decision, a bound on the block error rate is developed in Chapter 3 for use in the subsequent code evaluations.

Chapters 4, 5, 6 and 7 give brief descriptions of four different error correcting codes, then tell how they are evaluated by the Monte Carlo method. Attempts were made to develop a general evaluation technique for all block type codes, but the differences in the construction of the codes were too great to allow this development. Therefore, each code is analyzed and evaluated separately.

Chapter 8 presents the results of the computer programs for each of the codes under various conditions. The latter part of this chapter summarizes the conclusions drawn from the evaluations of the codes and development of the technique.

Some possible areas for further study are proposed in Chapter 9.

All programs were written in FORTRAN 63 (FORTRAN II), and used on a Control Data Corporation 1604 digital computer with the CO-OP operating system.

CHAPTER 2

MONTE CARLO METHOD

Random numbers can be used to assist in the solution of many problems. The main application of the method has been in the solution of statistical problems. In general, the method consists of making a selection of the random variables of the problem (or system), then evaluating typical outputs for each. Selections are repeated in order to obtain a meaningful sample. The statistics of the outputs are then examined in order to evaluate the actual problem.

Although the Monte Carlo technique is used to facilitate the solution of a problem, or the understanding of a problem, it must be noted that intelligent employment of the technique depends upon some understanding of the problem. In this investigation, considerable background study in coding was required for effective use of the method.

As a simple example of the application of the Monte Carlo Method, let us consider the probability of drawing a particular poker hand. We could, by use of well established methods, compute the figure analytically. Alternatively, we could use a Monte Carlo method to give us experimental results; we could draw a large number of hands, replacing the dealt cards and shuffling after each hand, and count how often the particular hand is dealt. The problem can be simulated by using a table of random numbers, with the numbers in the table assigned to the cards in such a way as to equally represent the cards. The probability of a particular poker hand could then be calculated from the results of a large number of groups of five draws from the table.

For such a simple problem, the method has little or no advantage, but for very involved problems and for those problems beyond the reach

of theoretical analysis, it can be quite useful. The advantages become even greater when high-speed digital computers are used to implement the problem. These methods have been widely used in studies of traffic congestion, switching circuits, war games, operations analysis, statistical mechanics, and other statistically oriented fields.

A less obvious application of the random sampling method is in the solution of deterministic problems; i.e., problems not directly involving probability. Many examples of the use of Monte Carlo methods to solve deterministic problems are cited in the literature (notably Hammersley and Handscomb [16]). Since we are concerned with probabilistic analyses in this paper, deterministic problem solutions will not be discussed except by noting that it is necessary to find a probabilistic analog that has a similar mathematical formulation.

It has become customary to refer to calculations involving random sampling as Monte Carlo methods. Such methods have been used for many years, but the real impetus for their use came during World War II when Ulam and Von Neumann used it to solve neutron diffusion problems.

As mentioned previously in the chapter, Monte Carlo methods may be used for solution of intractible problems. The neutron diffusion problem is representative of this type problem and its solution by Monte Carlo methods. This problem was too involved for an analytical solution and the alternative would have been expensive trial-and-error experimentation.

Monte Carlo methods are also used for problems with known results to facilitate an understanding of the internal mechanics of the way the random variables may be combined; to grasp the results of parameter variation; to gain more insight to the problem; or to verify the analytical results experimentally.

Random numbers can be derived from physical processes such as radioactive decay and thermionic noise. As an example of this, Rand Corporation has published a million digits produced by monitoring a random frequency source. Modern high speed computers may consume large quantities of random numbers at great rates. Under these conditions, reading them from storage devices becomes very limiting to the speed of computation. Consequently, strong emphasis has been placed on arithmetic generators since Von Neumann and Metropolis proposed their mid-square method about 1946.

Arithmetic generators are generally based upon some sort of recurrence relation involving integers. Each new number is generated from the previous one, as needed, by some sort of "scrambling" operation, so that the output is "randomly" drawn from the finite population of integers that the computer can produce. Some initial value is required to start the recurrence relation. After some number of recursions, a number that has already been produced will be repeated, thereby forming a closed loop sequence. The length of this loop is referred to as the period of the generator. The problem is to find a relation that produces a random sequence of numbers with a long period while using a minimum of computer time. These computer-generated numbers that pass statistical tests for randomness, even though they are produced by a deterministic process, are called psuedo-random numbers.

The method most widely used for psuedo-random number generation is the congruential, first proposed by Lehmer [17], and based upon the relation

$$x_{i+1} = ax_i \pmod{m}, \quad 0 \leq x_i \leq m \tag{2-1}$$

which means that the expression ax_i is to be divided by m, and x_{i+1} set equal to the remainder. Since the remainder is retained after division

by m, the period cannot be greater than m; therefore, m should be chosen to be a very large machine integer. With a suitable choice of constants in the congruence relation Eq. 2-1, very nearly the full period m can be achieved.

One might object that arithmetic generators, being completely deterministic, cannot produce truly random sequences, and should therefore not be used in Monte Carlo methods. The answer to this statement can be made in view of practical requirements: "true" randomness is not an essential requirement, as long as the statistical properties of the generator are sufficiently suitable to produce correct results for the problem being attacked.

As used for this paper, the uniform random number generator (FORTRAN CO-OP ID: G5 USNPGS RANF) used the congruence relation

$$R_{i+1} = R_i * 5^{17} \pmod{2^{36}} \qquad (2\text{-}2)$$

We know that the period will be very nearly equal to 2^{36}; therefore, we should be able to use the generator as random for nearly 2^{36} calls. The output was examined to detect any possible short term cyclic effects, but none were noted. It should also be pointed out that the use of the generator, in computer programs explained in subsequent chapters, to obtain known results is a most important test of its applicability to error correction analysis.

Psuedo-random numbers between zero and one were produced by the generator. Since the output is normalized to the unit interval, the probability that a number is in the interval $0 \leq x \leq x_o$ is equal to $P(x_o)$, and the probability that the number lies in the interval $x_o \leq x \leq 1$ is equal to $1 - P(x_o)$. We can therefore use the generator to simulate the production of binary digits with definite average bit error rates. Each psuedo-random

number, x, can be tested to determine if it is less than or greater than the probability of bit error, p. If $0 \leq x \leq p$, then an error can be assumed to have occurred; otherwise, no error is introduced. The result is a sequence of digit simulating signals with a known number of error indications. This sequence can then be fed to the decoder (or its simulation) in order to determine whether or not the errors can be corrected.

In the subsequent work, error correcting codes will be evaluated using the Monte Carlo method described above.

CHAPTER 3

CODING CAPABILITIES AND BOUNDS

3.1 Code Capability

Consider a binary channel with the symbols 0 and 1. Block codes use sequences of n channel symbols called n-tuples Certain selected n-tuples are used in error correction. The set of all n-tuples is a vector space. A selected set of such vectors is called a linear code if and only if it is a subspace of the space of all n-tuples.

The Hamming weight of a vector, v, denoted w(v), is defined as the number of nonzero components. We know that the Hamming distance between two vectors v_1 and v_2 is the number of positions in which they differ; therefore, the distance between v_1 and v_2 is $w(v_1 - v_2)$. If v_1 and v_2 are both code vectors of a linear code, then $v_1 - v_2$ must be a code vector, since the set of all code vectors is a vector space. Therefore the distance between any two code vectors equals the weight of some third code vector, and the minimum distance for a linear code equals the minimum weight of its nonzero vectors [1]. This property is useful in analyzing the bounds on error correction capabilities of codes.

For the purposes of this paper, the term capability of a code will define how many errors, t, a particular code may correct. If t or fewer errors occur in a block, then the block is correctly received after decoding. If more than t errors occur during transmission, then the block will be in error after decoding.

It is known that the probability of x errors occuring in a block of n digits is given by the binomial distribution:

$$P(x) = p^x q^{n-x} \binom{n}{x} \qquad (3\text{-}1)$$

The final block error rate may then be described by the cumulative binomial probability distribution where x is greater than the number of errors, t, that the code is capable of correcting. This relation may be abbreviated mathematically as

$$P(x>t) = \sum_{x=t+1}^{n} \binom{n}{x} p^x q^{n-x} \tag{3-2}$$

since t is an integer.

It is often useful to approximate the binomial distribution by the Poisson distribution. Lindgren [5] provides this approximation as

$$\lim_{\substack{n \to \infty \\ np \text{ fixed}}} \binom{n}{x} p^k q^{n-k} = e^{-np} \frac{(np)^k}{k!} \tag{3-3}$$

This approximation has been used in this paper where noted.

A bound on the block error rate for optimum codes will now be developed for use in evaluation of the results of the Monte Carlo testing. Binary codes are considered optimum if they are as effective as any other code with the same number of total symbols and the same number of information symbols. Those codes which meet the bounds described in Sections 3.2 and 3.3 are certainly optimum. The lower bound on a code is that bound which denotes the minimum probability of error after decoding. This bound on a code is usually expressed as a bound on the number of redundant bits, r, necessary to provide a particular distance or weight. (Sections 3.2 and 3.3). This bound can also be expressed in terms of the error rate after decoding. If a code is optimum, then the code's capability is maximized, and the error rate obtained is minimized. Therefore, when a code is optimum, the lower bound on the block error rate is given by Eq. 3-2. This property will be used, as applicable, in the succeding chapters.

3.2 Varsharmov-Gilbert Bound

Varsharmov [2] has refined Gilbert's bound and derived the following lower bound on the minimum distance for an error correction code. The theorem states that it is possible to construct a code of length n and minimum distance d with r parity-check symbols (and therefore at least k = n - r information symbols) where r is the smallest integer satisfying Eq. 3-4:

$$2^r \geq 1 + \sum_{i=1}^{d-2} \binom{n-1}{i} = \sum_{i=0}^{d-2} \binom{n-1}{i} \tag{3-4}$$

3.3 Hamming Bound

Hamming [3] derived an upper bound on the maximum minimum distance possible by the following method. Since an (n,k) linear code has 2^k code vectors and 2^{n-k} cosets (parity sequences), the number of vectors of weight m or less must be no greater than the number of cosets if the code is to correct all combinations of m or fewer errors. The theorem can be written as follows. Any n symbol code with minimum weight 2m + 1 or greater must have at least r check symbols where

$$r \geq \log_2 \left[1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{m} \right] \tag{3-5}$$

That is

$$2^r \geq 1 + \sum_{i=1}^{m} \binom{n}{i} = \sum_{i=0}^{m} \binom{n}{i} \tag{3-6}$$

The Hamming Bound applies to both linear and non-linear codes. Both the Hamming Bound and the Varsharmov-Gilbert Bound are plotted by Peterson [1].

CHAPTER 4

BINARY ERASURE CHANNEL ODD PARITY CODE

A simple error correction code will be analyzed first in order to clearly demonstrate the techniques used by the author.

Consider a phase-shift keyed (FSK) communication link as a binary erasure channel (BEC). Assume that the link is designed so that all signal levels except those representing 0 and 1 are locked out. If neither level is received during the time interval allotted, a blank or x is used as the output. The requirements for the BEC are now met.

The coding will use k information bits to allow transmission of 2^k different messages or words. The Hamming distance (H.D.) between these words is, of course, one. It is known that a Hamming distance of e + 1 is required to correct e erasures. If the capability of one erasure correction is desired, then a H.D. ≥ 2 is required.

In this example, one parity bit will be added to the k information bits to yield the required minimum distance. Figure 4-1 illustrates the procedure for k = 3. Note that the parity bit is added to give odd parity. With the added parity bit, the minimum Hamming distance becomes two, and there are n = k + 1 message bits. For this particular code, the redundancy (or rate at which information is transmitted) is given by the relation

$$R = \frac{m}{n} = \frac{m}{k + 1} \qquad (4\text{-}1)$$

k information bits			parity bit
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

n message bits

Figure 4-1. BEC Parity Code Example.

Since we now have the capability of correcting one error per block of n message bits, the expected block error rate after correction, PF, can be determined from the Eq. 3-2. The relation for **PF** is given in equation 4-2.

$$PF = P(X \geq 2) = \sum_{i=2}^{n} \binom{n}{i} p^{i}(1 - p)^{n-1} \qquad (4\text{-}2)$$

The program used to find PF by the Monte-Carlo technique is shown in Appendix A. The uniform random number generator was called upon to successively generate n numbers between 0 and 1. From the information in Chapter 2, it is known that $P(0 \leq x \leq p) = p$. As each random number, x, was generated, it was compared with p. If $x \leq p$, an error was assumed to have occurred. The number of errors was counted for each block, and if one error was found, the block was considered correctable. Similarly, the block was not correctable if the number of errors was two or greater. A loop was provided in the program in order to check a large number of blocks.

thereby insuring a meaningful statistical sample and a reasonable degree of statistical stability. The program generated 10,000 blocks for bit error probabilities, p, of 0.001 or 0.01, and 30,000 blocks for p = 0.0001. The results are shown in chapter 8. The number of correctable blocks is given by COR, and the number of uncorrected blocks is given by NON. The number of blocks with no errors is, of course, the total number of blocks generated minus COR and NON. The final block error rate, PF, was then computed by dividing NON by the number of block generating iterations. The expected, or theoretical, block error rates, PFTH, used for comparison were taken from Molina's tables of cumulative Poisson distribution [6].

Note that the program allows for the use of any reasonable p and n. Programs were written for varying p with n constant, and varying n while holding p constant.

The programs were used to find values of PF with n = 100 for p = 0.1, 0.01, 0.001 and 0.0001, and with p = 0.01 for n = 20, 40, 60, 80 and 100.

CHAPTER 5

SINGLE ERROR CORRECTING HAMMING CODE

Another single error correcting code will now be examined. The single error correcting Hamming code [4] is more sophisticated than the odd parity code for the binary erasure channel because it was developed for the binary symmetric channel. For the binary symmetric channel (BSC), the minimum Hamming distance (named for and invented by the same man who developed the code) required for the correction of t errors is 2t + 1. Therefore to correct one error we need a minimum distance of three.

Although Hamming codes are relatively simple and have the limited capability of single error correction, they have been widely used. The Bell System has made use of these codes for some time, and the Army's new tactical battlefield Digital Message Entry System uses one of the codes. Note that the redundancy becomes greater as the code length, n, becomes less; therefore, the error correction capability can be changed, in effect, by changing the length of the code. The results in Chapter 8 should clearly illustrate this point.

The Hamming codes are classified as linear, block, non-cyclic (although the devices used to perform the encoding and decoding may be designed as cyclic) parity check codes.

The code makes use of n bit blocks, m of which are information bits. The remaining n - m = k positions are used for parity check bits. These check bits are not necessarily the last k bits of the block; their placement is a function of the particular coding scheme used. A checking number may be determined and defined as follows:

a. Each of the parity check bits are computed in order as the code block of m information bits is received by the encoder.

b. After transmission and reception, the parity check bits are again computed from the code block entering the decoder.

c. Each newly computed parity check bit is compared with the corresponding check bit actually received.

d. If the two bits are the same, a 0 is entered in the checking number. If the two bits are different, a 1 is entered in the checking number.

e. We now have a binary checking number which is k bits long.

As long as the parity check bits are independent, any k positions in the code block may be selected for parity, and each check may function over any number of information bits.

The checking number is required to give the position of any single erroneous bit within the block in order to allow for the correction of that error. Since one error is n positions may occur in n different ways, and since there may be one sequence without error, the k-bit checking number must be able to distinguish between n + 1 different sequences. This places the following restriction on k:

$$2^k \geq n + 1 = m + k + 1 \qquad (5\text{-}1)$$

Note that for each k of two or greater, there are more than one m and n which fulfill equation 5-1. In the computer programs for the Hamming code, lengths of $n = 2^k - 1$ were used to provide maximum length and effectiveness while meeting the requirement of Eq. 5-1.

Let us now consider the bounds on the Hamming codes in order to find a basis for comparison with the values determined from the Monte Carlo program. The code is first compared with the Varsharmov-Gilbert lower bound.

Consider a code of length n = 7, determined from K = 3. As Hamming codes have minimum distance of three, Eq. 3.4 yields the following results:

$$2^r \geq 1 + \sum_{i=1}^{d-2} \binom{n-1}{i} = 1 + \binom{6}{1} = 7$$

therefore $r_{min} = 3$, which is also k. This means the code is optimum.

Similar results are found using the Hamming upper bound. Recall from Section 3.1 that the minimum weight of a linear code's nonzero vectors is equal to its minimum distance. This means that the weight of a Hamming code is always three. Now use will be made of Eq. 3-6 with m = 1, since the minimum weight, w, is given by w = 2m + 1 = 3. Note that this m is not the m used to signify the number of information bits. Eq. 3-6 becomes $2^r \geq 1 + n$, therefore $r_{min} = k$, and the Hamming bound is met for all Hamming codes. In view of this fact, we will use the tail of the cumulative binomial probability distribution, or its Poisson approximation, as both the bound on and expected value (PFTH) of the final block error rate, PF.

The program was designed to evaluate Hamming codes of maximum length; i.e., codes with

$$n = 2^k - 1$$

where k is an integer. As before, p, k, n and the number of block generating iterations are programmable to allow a wide selection of tests for evaluating different tests. As in the computer program for BEC Parity codes, n uniform random numbers between zero and one were successively generated and compared with p to simulate the transmission of both correct and incorrect binary digits. The number of errors per block were counted, then the block was classified as correct, correctable or uncorrectable.

The block error rate was again determined by dividing the number of uncorrectable blocks, NON, by the total number of blocks generated. The results shown in Section 8.2 were obtained by using a k initially set to three (for a block length of seven), then incremented after each run until a maximum value of k = 6 (for a block length of 63) was reached. The value of p was 0.1 throughout the evaluation. Again, it must be emphasized that short block lengths and high bit error probabilities were used in order to keep program running times fifteen minutes or shorter for a range of tests. The results are just as valid under these circumstances as they would be for longer lengths and lower bit error probabilities.

CHAPTER 6

GOLAY CODE

An example of an error correcting code that is more powerful than those evaluated in the preceeding chapters will now be studied. The Golay code is a fixed length code, designed for the binary symmetric channel, with 23 total bits, 12 of which are information positions.

Much use has been made of the Golay code since it is the only perfect, nontrivial code capable of correcting more than one error. The code is classified as linear, block, parity and cyclic. Since it is cyclic, it lends itself to implementation with shift registers. The above facts have made it popular, and its popularity has led to various implementations produced for the commercial market during the past five years. The code is somewhat limited by being fixed length, but its triple error correcting capacity allows it to operate effectively with high bit error rates. This capability should be evident from the results.

Golay [7, 8], in a search for perfect codes, found that

$$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2^{11}$$

and that this relationship indicated the possibility of a (23, 12) perfect binary code with the capability of correcting all patterns of three or fewer errors. Further study by Golay produced the code that he had predicted. The mechanics of the code will not be discussed in this paper because they are similar to those of the Bose-Chaudhuri codes which will be briefly discussed in the next chapter. Peterson [1] discusses the code and its generation using Galois fields. One implementation of the Golay code decoder is clearly and completely detailed by Rudolf [9], and the reader can use these references for an understanding of the workings of the code.

The bounds will now be discussed for the code. Table 6-1[1] lists the number of code vectors for the respective code weights of the (23, 12) code.

Weight	Number of Code Words of This Weight
0	1
7	253
8	506
11	1288
12	1288
15	506
16	253
23	1
24	0
	4096 Total

Table 6-1. Weight of Code Vectors in Golay Codes

Note that the total number of code words is $2^k = 2^{12} = 4096$. Since the code word with weight zero consists of all zeros, the minimum weight of a nonzero code vector is seven. From Chapter 3, we again use the relation that the minimum weight of a nonzero vector equals the minimum distance of the code. This means we must use an m equal to three in Eq. 3-6 to find the bound on the number of redundant bits, r. Substitution into Eq. 3-6 yields the following results:

$$2^r \geq \sum_{i=0}^{m} \binom{n}{i} = \binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3}$$

Note that this is the relation that led Golay to discover the code. Of course, r is the number of redundant bits, eleven.

The codes which have one information bit repeated 2m + 1 times will correct all the patterns of m or fewer errors. These trivial codes, the

[1]W. Wesley Peterson, Error-Correcting Codes (Massachusetts Institute of Technology: The M.I.T. Press; and New York: John Wiley and Sons, Inc.), p. 70.

Hamming codes, and the Golay code are the only known perfect codes for the binary symmetric channel.

Since the Golay code is perfect and therefore optimum, the expected value of the block error rate after correction will again be Eq. 3-2 with t equal 3; i.e., for the Golay (23, 12) code

$$PFTH = \sum_{x=4}^{n} \binom{n}{x} p^x q^{n-x} \qquad (6\text{-}1)$$

The Poisson distribution was not used to approximate PFTH in this case because the high bit error rate and relatively low block length combined to produce a poor approximation.

In this computer program, n and k are fixed, but p and the number of block generating iterations are programmable. Twenty-three random numbers were generated and tested to determine if they were larger or smaller than the value for p. As before, each number less than p was the analog of an error. The number of errors per block were counted, and if this number is one, two or three, the block was considered correctable; conversely, four or more errors meant an uncorrectable block. PF, NON, COR were computed by the same methods as before.

The most graphic results were obtained by varying p from 0.05 to 0.5 in increments of 0.05. This also allowed for 10,000 blocks to be generated, thereby providing a meaningful sample without exceeding the desired running time. These and other results are given in Chapter 8.

CHAPTER 7

BOSE-CHAUDHURI CODES

A variable length and variable error correcting ability code will now be evaluated. Bose and Chaudhuri [11, 12] discovered these codes in 1960. The codes are classified as linear, block, parity and cyclic. Since they are cyclic and, unlike most other known classes of codes, they cover a wide range in both rate and error-correcting capacity, their use has become widespread in the field of digital communications. The basic procedure for implementation of the codes can be accomplished very efficiently with a shift register, because the number of operations increases only as a small power of the length of the code.

The Bose-Chaudhuri codes are also known as Bose-Chaudhuri-Hocquenghem (BCH) codes since Hocquenghem [13] independently derived the codes about the same time as Bose and Chaudhuri.

Some of the organizations that have developed encoders and decoders for the code are General Electric, ITT-Communication Systems, International Business Machines, Inc., Honeywell, and Lincoln Laboratory. The U.S. Navy is presently testing one of these devices for use with its Fleet Broadcast System.

The construction of the BCH codes will now be briefly discussed. Some understanding of Galois or finite fields will be assumed in this presentation. An excellent source for this information is Bartee and Schneider's [14] article on computation with Galois fields. The article also includes a good method for logical design of Galois field arithmetic units.

Given an irreducible polynomial p(X) of degree m with the coefficients 0 and 1, a representation of the Galois field with 2^m elements, $GF(2^m)$,

can be formed. It consists of all polynomials of degree m - 1 or less. The polynomials can be added (modulo 2) term by term in the ordinary way. Multiplication is performed by multiplying in the ordinary way, reducing the answer modulo 2 and modulo p(X) to a polynomial of degree m - 1 or less. That is, consider p(X) = 0, then use this equation to eliminate terms of power greater than m - 1. It can be shown that certain of the polynomials, called primitive elements, have the property that the first $2^m - 1$ powers of such an element are exactly all the $2^m - 1$ nonzero field elements. Additionally, every nonzero field element is a root of the equation

$$X^{2^m-1} = 1$$

The converse also holds; thus, if α is any element of the field,

$$\alpha^{-1} = \alpha^{2^m-2}$$

The field elements may also be thought of as vectors whose components are the coefficients of the polynomials. The sum of the two vectors corresponds to the sum of the corresponding polynomials.

The Bose-Chaudhuri codes are described by giving the matrix of parity check rules, which is the matrix given by equation 7-1, where α is a primitive element of the field.

$$M = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha & \alpha^3 & \cdots & \alpha^{2t-1} \\ \alpha^2 & (\alpha^3)^2 & \cdots & (\alpha^{2t-1})^2 \\ \cdot & \cdot & \cdots & \\ \cdot & \cdot & \cdots & \\ \cdot & \cdot & \cdots & \\ \alpha^{2^m-2} & (\alpha^3)^{2^m-2} & \cdots & (\alpha^{2t-1})^{2^m-2} \end{bmatrix} \qquad (7\text{-}1)$$

This is a $(2^m - 1) \times t$ matrix of $GF(2^m)$ elements, but by considering each field element as a vector of m binary digits, M is a $(2^m - 1) \times mt$ matrix of binary digits. A vector of $2^m - 1$ bits is considered a code word if it satisfies the parity check described by each column; i.e., if the product of this vector with the matrix is zero. In other words, the set of all code words is the (left) null space of the matrix, M.

Note that the rank $(M) \leq mt$. Since there is only one Galois field $(GF2^m)$, this rank is a definite function of m and t and will be denoted by R(m,t). When $R(m,t) < mt$, we can choose R(m,t) independent columns of M, and delete the other columns dependent on them. Bose and Ray-Chaudhuri use the above properties to arrive at their basic theorem which is stated as follows:

If $n = 2^m - 1$, we can obtain a t-error correcting (n,k) binary group code where

$$k = 2^m - 1 - R(m,t) \geq 2^m - 1 - mt \qquad (7\text{-}2)$$

or

$$k = n - R(m,t) \geq n - mt, \qquad (7\text{-}3)$$

and mt is the number of parity bits.

Table 7-1 shows the BCH codes of length 63 or less.[2] Note that k is not equal to n - mt for the (31, 11), (31, 6), (63, 18), (63, 16), (63, 10) and (63, 7) codes. This means that $R(m,t) \neq mt$ for these codes.

A detailed explanation of the codes can be obtained from Bose and Chaudhuri's original paper [11]. Another class of BCH codes can be found by letting α be a nonprimitive root of $GF(2^m)$. These codes are derived in Bose and Ray-Chaudhuri's second paper [12].

The bounds on the BCH codes will now be examined in order to find

[2]Peterson, op. cit., p. 166.

n	k	t	n	k	t
7	4	1	31	6	7
15	11	1	63	57	1
	7	2		51	2
	5	3		45	3
31	26	1		39	4
	21	2		36	5
	16	3		30	6
	11	5		24	7

Table 7-1. Bose-Chaudhuri Codes Generated by Primitive Elements of Order Less than 2^6.

bases of comparison for the results produced by the Monte Carlo computer programs. The Bose-Chaudhuri codes are a generalization of the Hamming codes; the case t = 1 gives the Hamming code for each $n = 2^m - 1$. From Chapter 5, we know that these codes are perfect, and therefore optimum. In addition, Gorenstein, Peterson and Zierler [15] have shown that all BCH double-error-correcting codes are quasi-perfect and hence optimum. Peterson [1] has shown that all codes of length 15 or less are optimum. Since these three cases are optimum, we know that we can use the tail of the binomial distribution, Eq. 3-2, as our bound on the final block error rate.

Programs were written to evaluate the BCH codes of length 63 or less with a capacity for correction of up to eight errors; therefore, we need some bound for the 31 and 63 digit codes with t greater than two.

Consider the (31, 16) triple-error correcting code. The use of the Varsharmov-Gilbert bound, Eq. 3-4, yields

$$2^r \geq 1 + \binom{31}{1} + \binom{31}{2} + \binom{31}{3} + \binom{31}{4} + \binom{31}{5}$$

since $d = 2t + 1 = 7$ and $d - 2 = 5$. The result is $2^r \geq 25{,}824$, and the minimum r which satisfies this relation is $r_b = 15$. But the number of redundant bits used by the code is $n - k = 15$; therefore, the code meets the bound and we can again use Eq. 3-2 to predict PFTH. In similar computations for the (31, 11) and (31, 6) codes, as well as for the 3, 4, 5, 6 and 7-error correcting 63 digit codes, r_b was found to be equal to or less than the number of redundant digits actually used in the code. Peterson [1] extended this analysis to show that codes of even length 1023 ($2^{10} - 1$) are approximately at the Varsharmov-Gilbert bound and hence are certainly good. He concluded that any non-optimum behavior these codes may have occurs only in codes so long that there is no hope of demonstrating it in practical devices. We can therefore use Eq. 3-2 to determine PFTH for each of the codes analyzed.

Four similar programs were used to obtain results for the range $7 \leq n \leq 63$ with $1 \leq t \leq 7$. Use of the proper m (which determines n), k and t will extend the use of the programs for any BCH code. The listings of these programs are in appendix A.

The first program is for $7 \leq n \leq 31$ with $1 \leq t \leq 5$. Within this range, the defining equations for the code are

$$n = 2^m - 1 \ ,$$

$$M = mt \ ,$$

$$\text{and } k = n - M \ ,$$

where n = number of bits in block, M = number of parity bits, and k = number of information positions in the block. The information rate in this case is k/n. Rather than attempting to explain the program in detail, the flowchart of the program, Figure 7-1, shows the logic used. Initial values of $t = 1$ and $m = 3$, and terminal values of $t_{max} = 3$ and

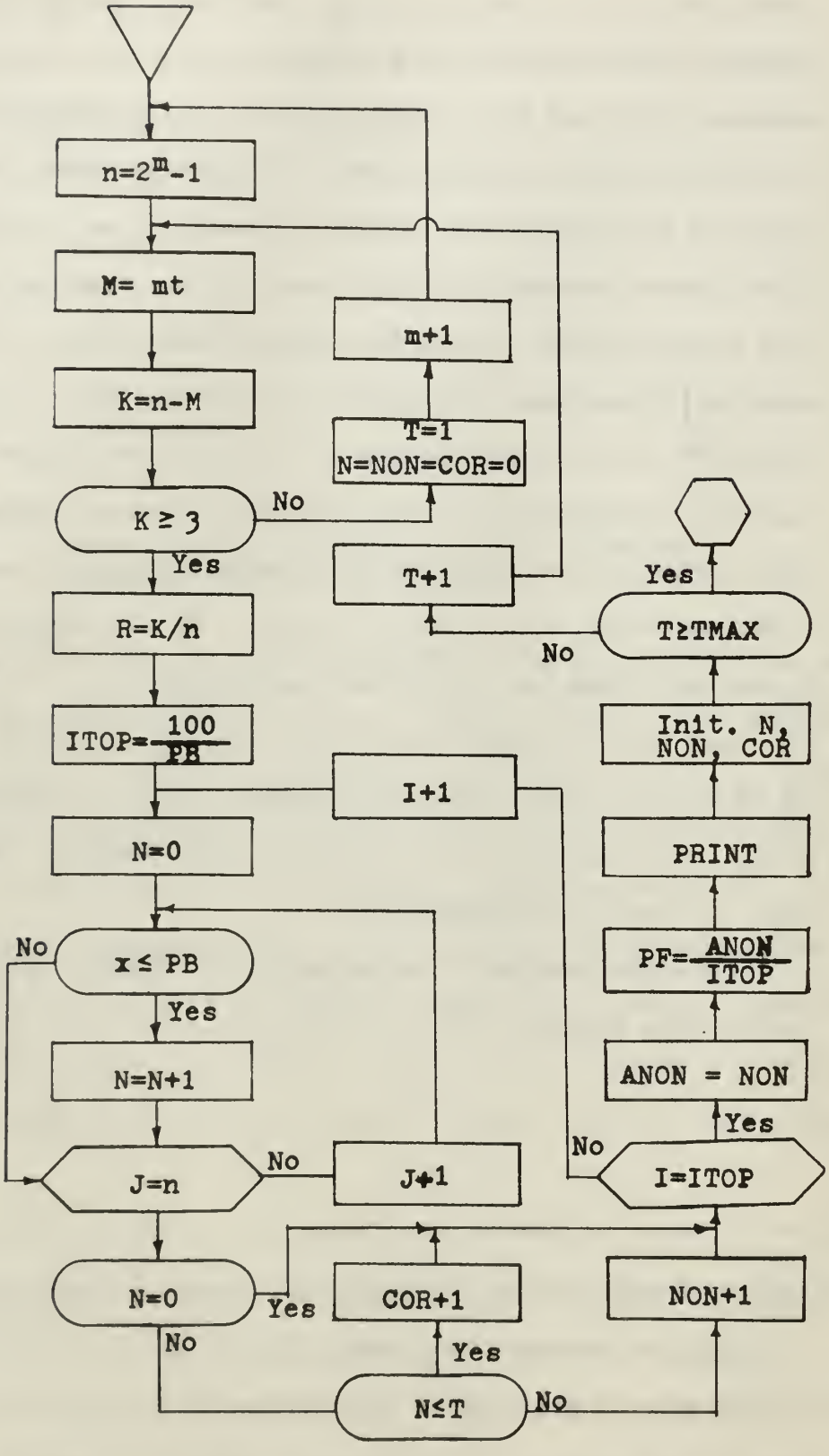

Fig. 7-1. Flowchart for Basic Bose-Chaudhuri Program.

m = 5 were inserted to evaluate the results over the desired range. As in the previous programs, the random number, x, was compared to p to determine if an error had occurred. The number of errors per block was counted and then compared with the capacity, at that time, of the code to correct errors. If the number of errors per block, N, was greater than t, the block was uncorrectable. The final block error rate, PF, was computed as before. Good results were obtained with p = 0.1 and 1000 blocks.

The program was modified by removing the test for $k \geq 3$, and the incrementing of m to evaluate the codes in the range $1 \leq t \leq 7$ for a block length of 63. The program thus modified used a constant m = 6, p = 0.1 and 5000 block iterations.

Whenever the rank of the matrix of parity check rules (eq. 7-1) yields

$$R(m,t) \neq mt \ ,$$

a program similar to that for the Golay code must be used. Values for n, k (and therefore M), and t are chosen from use of the rank of the matrix; or as in this case, by the use of a table such as Table 7-1. The values of n, k and t for the (31, 11) and (31, 6) code were chosen using the table, and inserted in the modified Golay program. The correctability, final block error rate and redundancy are determined as in the other programs.

CHAPTER 8

RESULTS AND CONCLUSIONS

8.1 BEC Parity Results.

Table 8-1 lists the theoretical block error rates (PFTH) for the bit error rates (PB) and block length (NB) shown. The values of PFTH are from Molina [5].

NB	PB	PFTH
100	0.1	0.999501
100	0.01	0.264241
100	0.001	0.004679
100	0.0001	0.000049
20	0.01	0.017523
40	0.01	0.061552
60	0.01	0.121901
80	0.01	0.191208

Table 8-1. Block Error Rates for BEC Parity Code.

Experimental values of block error rate (PF), obtained by use of the computer programs of Chapter IV, are shown on Tables 8-2 and 8-3. As stated before,the program was run with NB = 100 while varying PB, and with PB = 0.01 while varying NB. Graphical presentation of the results was considered, but since the experimental results differ from the expected values by so little, data points would be virtually on top of the expected positions, as well as each other. The same thing would be true for a plot of the logarithms of PFTH versus PB or NB, since the errors are rarely greater than about 0.004.

These results lead us to conclude that the method used does indeed

```
PB=1.000E-02    NB=      20    COR=    1622
NCN=     171    PF=1.7100E-02

PB=1.000E-02    NB=      40    COR=    2678
NCN=     639    PF=6.3900E-02

PB=1.000E-02    NB=      60    COR=    3355
NCN=    1192    PF=1.1920E-01

PB=1.000E-02    NB=      80    COR=    3582
NCN=    1899    PF=1.8990E-01

PB=1.000E-02    NB=     100    COR=    3638
NCN=    2696    PF=2.6960E-01

PB=1.000E-02    NB=      20    COR=    1664
NCN=     155    PF=1.5500E-02

PB=1.000E-02    NB=      40    COR=    2686
NCN=     592    PF=5.9200E-02

PB=1.000E-02    NB=      60    COR=    3266
NCN=    1248    PF=1.2480E-01

PB=1.000E-02    NB=      80    COR=    3684
NCN=    1897    PF=1.8970E-01

PB=1.000E-02    NB=     100    COR=    3700
NCN=    2610    PF=2.6100E-01

PB=1.000E-02    NB=      20    COR=    1665
NCN=     187    PF=1.8700E-02

PB=1.000E-02    NB=      40    COR=    2622
NCN=     618    PF=6.1800E-02
```

Table 8-2. Results for BEC Parity Code with Varying Block Length.

PB=1.000E-01	NB=	100	COR=	1	NON=	9998	PF=9.9980E-01
PB=1.000E-02	NB=	100	COR=	3766	NON=	2664	PF=2.6640E-01
PB=1.000E-03	NB=	100	COR=	860	NON=	55	PF=5.5000E-03
PB=1.000E-01	NB=	100	COR=	2	NON=	9998	PF=9.9980E-01
PB=1.000E-02	NB=	100	COR=	3703	NON=	2643	PF=2.6430E-01
PB=1.000E-03	NB=	100	COR=	900	NON=	48	PF=4.8000E-03
PB=1.000E-01	NB=	100	COR=	6	NON=	9994	PF=9.9940E-01
PB=1.000E-02	NB=	100	COR=	3764	NON=	2685	PF=2.6850E-01
PB=1.000E-03	NB=	100	COR=	902	NON=	54	PF=5.4000E-03

Table 8.3. BEC Parity Results for Varying Error Rate.

accurately estimate the bounds on the block error rate for this relatively simple code.

8.2 Hamming Results.

Table 8-4 provides the expected block error rates for various Hamming Codes. These result from the use of Eqs. 5-1 and 3-2.

NB	PB	PFTH
7	0.1	0.149694
15	0.1	0.450957
31	0.1	0.830435
63	0.1	0.986595

Table 8-4. Block Error Rates for Hamming Codes.

Four sets of experimental values of block error rate (PF), as listed by the CDC 1604, are shown in Table 8-5. Note that the maximum difference between PF and PFTH is about 0.012, and the difference is usually about 0.005. In each case, 5000 blocks were generated and tested. As expected, the number of correctable blocks declines, and the number of uncorrectable blocks increases, as the block length increases. This is the expected result since both the average frequency at which errors are produced, and the error correcting capability of the codes are constant. This variation in PF illustrates the use of the Hamming code's variable block length to provide the desired error correcting rate even though each code can only correct one error per block. The same effect can be observed for the BEC Parity Code.

Again, the results of the Monte Carlo program indicate the validity of its use to determine bounds on the error correction rate of a particular code.

```
PB=1.000E-01   NB=     7   COR= 1903   NON=  755
RD=5.714E-01   PF=1.5100E-01

PB=1.000E-01   NB=    15   COR= 1709   NON= 2214
RD=7.333E-01   PF=4.4280E-01

PB=1.000E-01   NB=    31   COR=  682   NON= 4136
RD=8.387E-01   PF=8.2720E-01

PB=1.000E-01   NB=    63   COR=   50   NON= 4947
RD=9.048E-01   PF=9.8840E-01

PB=1.000E-01   NB=     7   COR= 1834   NON=  751
RD=5.714E-01   PF=1.5020E-01

PB=1.000E-01   NB=    15   COR= 1711   NON= 2205
RD=7.333E-01   PF=4.4100E-01

PB=1.000E-01   NB=    31   COR=  641   NON= 4148
RD=8.387E-01   PF=8.2960E-01

PB=1.000E-01   NB=    63   COR=   61   NON= 4933
RD=9.048E-01   PF=9.8660E-01

PB=1.000E-01   NB=     7   COR= 1918   NON=  725
RD=5.714E-01   PF=1.4500E-01

PB=1.000E-01   NB=    15   COR= 1680   NON= 2276
RD=7.333E-01   PF=4.5520E-01

PB=1.000E-01   NB=    31   COR=  658   NON= 4138
RD=8.387E-01   PF=8.2760E-01

PB=1.000E-01   NB=    63   COR=   38   NON= 4959
RD=9.048E-01   PF=9.9180E-01

PB=1.000E-01   NB=     7   COR= 1911   NON=  723
RD=5.714E-01   PF=1.4460E-01

PB=1.000E-01   NB=    15   COR= 1811   NON= 2164
RD=7.333E-01   PF=4.3280E-01

PB=1.000E-01   NB=    31   COR=  655   NON= 4154
RD=8.387E-01   PF=8.3080E-01

PB=1.000E-01   NB=    63   COR=   40   NON= 4953
RD=9.048E-01   PF=9.9060E-01
```

Table 8-5. Hamming Code Results.

8.3 Golay Results.

The theoretical block error rates, after correction, for the perfect Golay (23, 12) code are shown in Table 8-6. As stated earlier, the values are the results of Eq. 6-1, as published in the NBS tables [10].

PB	PFTH
0.01	0.000076
0.05	0.025814
0.10	0.192731
0.15	0.460370
0.20	0.703469
0.25	0.863043
0.30	0.946156
0.35	0.981920
0.40	0.994839
0.45	0.998763
0.50	0.999756

Table 8-6. Block Error Rates for Golay Code.

High bit error rates were used to obtain a range with a good variation in both dependent and independent variables, while at the same time using reasonable running times to provide good statistical stability. It would have been desireable to provide more block generating iterations for p = PB = 0.45 and 0.50, but the trend is clear, even if the results are not completely valid (statistically stable) for these two points. Note that PFTH equals 7.6×10^{-5} for PB equal to 10^{-2}. Even more indicative of the power of the Golay code is that PFTH is less than 10^{-7} for PB of 10^{-3}. This large capability of the code to correct errors is the

PB=5.000E-02	NB=	23	COR=	3340	NON=	115	PF=2.3000E-02
PB=1.000E-01	NB=	23	COR=	3573	NON=	977	PF=1.9540E-01
PB=1.500E-01	NB=	23	COR=	2563	NON=	2315	PF=4.6300E-01
PB=2.000E-01	NB=	23	COR=	1437	NON=	3520	PF=7.0400E-01
PB=2.500E-01	NB=	23	COR=	654	NON=	4338	PF=8.6760E-01
PB=3.000E-01	NB=	23	COR=	286	NON=	4713	PF=9.4260E-01
PB=3.500E-01	NB=	23	COR=	110	NON=	4890	PF=9.7800E-01
PB=4.000E-01	NB=	23	COR=	22	NON=	4978	PF=9.9560E-01
PB=4.500E-01	NB=	23	COR=	9	NON=	4991	PF=9.9820E-01
PB=5.000E-01	NB=	23	COR=	2	NON=	4998	PF=9.9960E-01

Table 8-7. Golay Code Results.

PB=5.000E-02	NB=	23	COR=	3334	NON=	141	PF=2.8200E-02
PB=1.000E-01	NB=	23	COR=	3574	NON=	996	PF=1.9920E-01
PB=1.500E-01	NB=	23	COR=	2587	NON=	2291	PF=4.5820E-01
PB=2.000E-01	NB=	23	COR=	1451	NON=	3521	PF=7.0420E-01
PB=2.500E-01	NB=	23	COR=	663	NON=	4329	PF=8.6580E-01
PB=3.000E-01	NB=	23	COR=	286	NON=	4710	PF=9.4200E-01
PB=3.500E-01	NB=	23	COR=	86	NON=	4914	PF=9.8280E-01
PB=4.000E-01	NB=	23	COR=	27	NON=	4973	PF=9.9460E-01
PB=4.500E-01	NB=	23	COR=	5	NON=	4995	PF=9.9900E-01
PB=5.000E-01	NB=	23	COR=	2	NON=	4998	PF=9.9960E-01

Table 8-7. Continued

PB=5.000E-02	NB=	23	COR=	3329	NON=	124	PF=2.4800E-02
PB=1.000E-01	NB=	23	COR=	3582	NON=	934	PF=1.8680E-01
PB=1.500E-01	NB=	23	COR=	2561	NON=	2331	PF=4.6620E-01
PB=2.000E-01	NB=	23	COR=	1467	NON=	3504	PF=7.0080E-01
PB=2.500E-01	NB=	23	COR=	697	NON=	4299	PF=8.5980E-01
PB=3.000E-01	NB=	23	COR=	281	NON=	4717	PF=9.4340E-01
PB=3.500E-01	NB=	23	COR=	108	NON=	4891	PF=9.7820E-01
PB=4.000E-01	NB=	23	COR=	29	NON=	4971	PF=9.9420E-01
PB=4.500E-01	NB=	23	COR=	9	NON=	4991	PF=9.9820E-01
PB=5.000E-01	NB=	23	COR=	0	NON=	5000	PF=1.0000E 00

Table 8-7. Continued

reason for the high bit error rates for the Monte Carlo experiment results listed in Table 8-7; otherwise, the number of blocks required to produce stable results would be quite high.

Note the variations in the number of correctable, COR, and the number of uncorrectable, NON, blocks. From the results shown in Table 8-7, and other results not shown in this paper, it was learned that COR increased as PB was increased from very small values to about 0.1, then decreased as PB continued to increase. At the same time, NON was, of course, increasing then decreasing for the same variation in PB. This effect comes from the fact that for low values of PB, no errors are produced for many blocks. As PB increases, the number of blocks with one to three errors, and thus COR, increase rapidly, while occasionally four or more errors occur leading to the slower rise in NON. As PB approaches 4/23; i.e., that value of PB for which an average of four errors are generated, COR reaches its maximum. This maximum evidently occurs at about PB equal 0.1. The number of uncorrectable blocks increases at a rapid rate near this transition point, then the rate slows as nearly all the blocks begin to contain four or more errors.

The differences between PF and PFTH are again extremely small; at most 0.007 for the three runs shown. The validity of using the Monte Carlo method to simulate the Golay code has been amply demonstrated by these results.

8.4 Bose-Chaudhuri Results.

Results for the Monte Carlo experiments using computer programs for the Bose-Chaudhuri codes are presented below. Table 8-8 includes the final block error rates for particular block lengths, information bits, error capability and bit error rates.

The high-speed line printer listings for the programs are shown on

tables 8-9, 10, 11 and 12. The result for BCH codes of length $7 \leq n \leq 31$ and correction capability $1 \leq t \leq 3$ are shown in Table 8-9. The results are not as extremely accurate as those for the preceeding codes, but they are quite good. The maximum discrepancy is about 0.036, while the average difference is only about 0.009 for six-place decimals. With the exception of these differences, the other results (the variations in NON, COR and PF with changes in NB, KB and T) are as expected.

Table 8-10, which gives results for the BCH (31, 11) five error correcting code, reveals excellent agreement between PFTH (0.083421) and

NB	KB	T	PB	PFTH
7	4	1	0.1	0.149694
15	11	1	0.1	0.450957
15	7	2	0.1	0.184061
15	5	3	0.1	0.055556
31	26	1	0.1	0.830435
31	21	2	0.1	0.611414
31	16	3	0.1	0.376170
31	11	5	0.1	0.083421
31	6	7	0.1	0.009588
63	57	1	0.1	0.986595
63	51	2	0.1	0.950154
63	45	3	0.1	0.873626
63	39	4	0.1	0.753096
63	36	5	0.1	0.601228
63	30	6	0.1	0.441767
63	24	7	0.1	0.298252

Table 8-8. Block Error Rates for Bose-Chaudhuri Codes.

```
 PB=1.000E-01   NB=     7   KB=     4   T=     1
CCR=   389   NON=   138   RD=5.714E-01   PF=1.3814E-01

 PB=1.000E-01   NB=    15   KB=    11   T=     1
CCR=   340   NON=   486   RD=7.333E-01   PF=4.8649E-01

 PB=1.000E-01   NB=    15   KB=     7   T=     2
CCR=   612   NON=   178   RD=4.667E-01   PF=1.7818E-01

 PB=1.000E-01   NB=    15   KB=     3   T=     3
CCR=   724   NON=    60   RD=2.000E-01   PF=6.0060E-02

 PB=1.000E-01   NB=    31   KB=    26   T=     1
CCR=   141   NON=   816   RD=8.387E-01   PF=8.1682E-01

 PB=1.000E-01   NB=    31   KB=    21   T=     2
CCR=   349   NON=   609   RD=6.774E-01   PF=6.0961E-01

 PB=1.000E-01   NB=    31   KB=    16   T=     3
CCR=   579   NON=   383   RD=5.161E-01   PF=3.8338E-01



 PB=1.000E-01   NB=     7   KB=     4   T=     1
CCR=   361   NON=   154   RD=5.714E-01   PF=1.5415E-01

 PB=1.000E-01   NB=    15   KB=    11   T=     1
CCR=   316   NON=   448   RD=7.333E-01   PF=4.4845E-01

 PB=1.000E-01   NB=    15   KB=     7   T=     2
CCR=   581   NON=   171   RD=4.667E-01   PF=1.7117E-01

 PB=1.000E-01   NB=    15   KB=     3   T=     3
CCR=   736   NON=    60   RD=2.000E-01   PF=6.0060E-02

 PB=1.000E-01   NB=    31   KB=    26   T=     1
CCR=   133   NON=   829   RD=8.387E-01   PF=8.2983E-01

 PB=1.000E-01   NB=    31   KB=    21   T=     2
CCR=   364   NON=   602   RD=6.774E-01   PF=6.0260E-01

 PB=1.000E-01   NB=    31   KB=    16   T=     3
CCR=   602   NON=   361   RD=5.161E-01   PF=3.6136E-01
```

Table 8-9. BCH Results for $1 \leq N \leq 31$ and $1 \leq T \leq 3$

```
 PB=1.000E-01    NB=     7    KB=     4    T=     1
CCR=   366    NON=   155    RD=5.714E-01    PF=1.5516E-01

 PB=1.000E-01    NB=    15    KB=    11    T=     1
CCR=   366    NON=   430    RD=7.333E-01    PF=4.3043E-01

 PB=1.000E-01    NB=    15    KB=     7    T=     2
CCR=   610    NON=   183    RD=4.667E-01    PF=1.8318E-01

 PB=1.000E-01    NB=    15    KB=     3    T=     3
CCR=   746    NON=    56    RD=2.000E-01    PF=5.6056E-02

 PB=1.000E-01    NB=    31    KB=    26    T=     1
CCR=   128    NON=   828    RD=8.387E-01    PF=8.2883E-01

 PB=1.000E-01    NB=    31    KB=    21    T=     2
CCR=   347    NON=   604    RD=6.774E-01    PF=6.0460E-01

 PB=1.000E-01    NB=    31    KB=    16    T=     3
CCR=   590    NON=   359    RD=5.161E-01    PF=3.5936E-01
```

Table 8-9. Continued

```
PB=1.000E-01   NB=   31   KB=   11   T=    5
COR=  884   NON=   81   RD=3.548E-01   PF=8.1081E-02

PB=1.000E-01   NB=   31   KB=   11   T=    5
COR=  879   NON=   78   RD=3.548E-01   PF=7.8078E-02

PB=1.000E-01   NB=   31   KB=   11   T=    5
COR=  872   NON=   88   RD=3.548E-01   PF=8.8088E-02
```

Table 8-10. BCH (31,11) Code Results.

```
PB=1.000E-01   NB=   31   KB=    6   T=    7
COR= 9516   NON=   91   RD=1.935E-01   PF=9.1000E-03

PB=1.000E-01   NB=   31   KB=    6   T=    7
COR= 9524   NON=   87   RD=1.935E-01   PF=8.7000E-03

PB=1.000E-01   NB=   31   KB=    6   T=    7
COR= 9489   NON=   89   RD=1.935E-01   PF=8.9000E-03
```

Table 8-11. BCH (31,6) Code Results.

```
PB=1.000E-01   NB=    63   KB=    57   T=     1
COR=     6   NON=   991   RD=9.048E-01   PF=9.9199E-01

PB=1.000E-01   NB=    63   KB=    51   T=     2
COR=    52   NON=   946   RD=8.095E-01   PF=9.4695E-01

PB=1.000E-01   NB=    63   KB=    45   T=     3
COR=   115   NON=   883   RD=7.143E-01   PF=8.8388E-01

PB=1.000E-01   NB=    63   KB=    39   T=     4
COR=   222   NON=   775   RD=6.190E-01   PF=7.7578E-01

PB=1.000E-01   NB=    63   KB=    33   T=     5
COR=   396   NON=   600   RD=5.238E-01   PF=6.0060E-01

PB=1.000E-01   NB=    63   KB=    27   T=     6
COR=   548   NON=   449   RD=4.286E-01   PF=4.4945E-01

PB=1.000E-01   NB=    63   KB=    21   T=     7
COR=   717   NON=   280   RD=3.333E-01   PF=2.8028E-01
```

```
PB=1.000E-01   NB=    63   KB=    57   T=     1
COR=    11   NON=   985   RD=9.048E-01   PF=9.8599E-01

PB=1.000E-01   NB=    63   KB=    51   T=     2
COR=    39   NON=   960   RD=8.095E-01   PF=9.6096E-01

PB=1.000E-01   NB=    63   KB=    45   T=     3
COR=   102   NON=   896   RD=7.143E-01   PF=8.9690E-01

PB=1.000E-01   NB=    63   KB=    39   T=     4
COR=   225   NON=   770   RD=6.190E-01   PF=7.7077E-01

PB=1.000E-01   NB=    63   KB=    33   T=     5
COR=   374   NON=   623   RD=5.238E-01   PF=6.2362E-01

PB=1.000E-01   NB=    63   KB=    27   T=     6
COR=   536   NON=   462   RD=4.286E-01   PF=4.6246E-01

PB=1.000E-01   NB=    63   KB=    21   T=     7
COR=   732   NON=   264   RD=3.333E-01   PF=2.6426E-01
```

Table 8-12. BCH Code Results for $n \leq 63$ and $1 \leq T \leq 7$.

```
PB=1.000E-01   NB=    63   KB=    57   T=     1
COR=     9   NON=   988   RD=9.048E-01   PF=9.8899E-01

PB=1.000E-01   NB=    63   KB=    51   T=     2
COR=    40   NON=   957   RD=8.095E-01   PF=9.5796E-01

PB=1.000E-01   NB=    63   KB=    45   T=     3
COR=   104   NON=   895   RD=7.143E-01   PF=8.9590E-01

PB=1.000E-01   NB=    63   KB=    39   T=     4
COR=   224   NON=   774   RD=6.190E-01   PF=7.7477E-01

PB=1.000E-01   NB=    63   KB=    33   T=     5
COR=   394   NON=   605   RD=5.238E-01   PF=6.0561E-01

PB=1.000E-01   NB=    63   KB=    27   T=     6
COR=   550   NON=   449   RD=4.286E-01   PF=4.4945E-01

PB=1.000E-01   NB=    63   KB=    21   T=     7
COR=   698   NON=   298   RD=3.333E-01   PF=2.9830E-01
```

Table 8-12. Continued

the three experimental values of PF. The error is no more than about 0.0053 thereby justifying, again, our use of the Monte Carlo method for this problem.

Similarly, experimental results for the BCH (31, 6) code differ very slightly from the theoretical values. The number of blocks generated had to be increased to 10,000 in order to obtain a reasonably stable number of uncorrectable blocks. From the expected block error rate of 0.009588, we know that an average of about 96 uncorrectable blocks should be generated per 10,000 blocks; whereas 91, 87 and 89 were actually generated for the three trials shown. Somewhat better results would probably be obtained by the use of more iterations.

The program for the BCH 63-bit long codes produced the results shown in Table 8-12. Differences between PF and PFTH for three runs with these codes averaged about 0.012, with a maximum of approximately 0.034. The variations in NON and COR are as expected.

Although the results are not as good for the Bose-Chaudhuri codes, they are still more than adequate to verify the correctness of the use of the Monte Carlo method for evaluation of BCH codes.

8.5 Conclusions.

We have described the use of the Monte Carlo technique for evaluation of error correcting codes. The results obtained were compared to the bounds, developed in Chapter 3, on the block error rates after use of the codes. The close agreement between the theoretical and experimental values for block error rates after correction must establish confidence in the Monte Carlo method's use to determine error correction bounds and capabilities. Since the method produced such good results for codes with well defined bounds, it can be expected that the use of the method could be extended to codes which do not have readily determined bounds. In such an application, results obtained by the Monte Carlo method might assist in the formulation of the bounds. This task is recommended in Chapter 9, as part of a possible thesis area. This method could also be used to verify coding bounds that may have been determined analytically, but need more collaboration before their more complete acceptance.

The programs could easily be modified to determine bit error rates after correction by counting the total number of uncorrectable errors for each run. This number would then be divided by the total number of bits generated to obtain the final bit error rate. This cannot be done in an application, however, because error correcting devices do not give an indication of how many errors may be in a block which had more errors than the code was capable of correcting. The device can be made to indicate that more errors have occurred than it is capable of correcting.

Another modification of the programs could be made to reduce the running time. Suppose low bit error rates and long block lengths were desired. The capability of the code could be used to determine, analytically, the probability p' of more than t errors per block. The random number stream could be checked to determine if x was less than this new

probability p'; if so, then an error could be assumed to have occurred. The program could then compute PF, COR, NON, etc. as before. The running time for the simulation would thus be reduced by a factor approximately equal to the block length, n.

A very important use of the technique developed here could be the simulation of a data stream with errors. The errors could be produced in the following manner: each random number could be tested to see if it lies in the interval $0 \leq x \leq p$; if it does, then the corresponding bit is changed to the opposite binary level. The original data stream would then be fed into the encoder for parity addition and manipulation to produce the encoded stream. The encoded stream would then be modulated by the Monte Carlo sequence to simulate errors before entry into the decoder. The output of the decoder could then be compared with the original data to determine the final error rate.

CHAPTER 9

RECOMMENDATIONS

The following tasks are proposed for further thesis study:

1. Evaluation of bounds on other codes for the binary erasure channel.

2. Evaluation of bounds on convolutional type codes for both the BEC and BSC.

3. Evaluation of bounds and simulation of burst error correcting codes.

4. Evaluation of bounds for code(s) with analytically undefined bounds.

BIBLIOGRAPHY

1. Peterson, W.W., Error-Correcting Codes, Massachusetts Institute of Technology: The M.I.T. Press, 1961.

2. Varsharmov, R.R., "Estimate of the Number of Signals in Error Correcting Codes," Doklady A.N.S.S.R., 117, No. 5, 739-741 (1957).

3. Hamming, R.W., "Error Detecting and Error Correcting Codes," Bell System Technical Journal, 29, 174-160 (1950).

4. Hamming, R.W., "Error Detecting and Error Correcting Codes," Bell System Technical Journal, 26, 147-160 (1950).

5. Lindgren, B.W., Statistical Theory, New York: The MacMillan Company, 1961.

6. Molina, E.C., Poisson's Exponential Binomial Limit, New York: D. Van Nostrand Co., 1949.

7. Golay, M.J.E., "Notes on Digital Coding," Proceedings of the I.R.E., 37, Correspondence, 657 (1949).

8. Golay, M.J.E., "Binary Coding," I.R.E. Transactions, PGIT-4, 23-28 (1954).

9. Rudolf, L., "Easily Implemented Error Correction Encoding-Decoding," Technical Report 62MCD2, General Electric Co., Oklahoma City, Oklahoma.

10. Tables of Binomial Probability Distribution, Applied Mathematics Series No. 6, National Bureau of Standards, Washington, D.C., 1949.

11. Bose, R.C., and Ray-Chaudhuri, D.K., "on a Class of Error Correcting Ginary Group Codes," Information and Control, 3, 68-79 (1960).

12. Bose, R.C., and Ray-Chaudhuri, D.K., "Further Results on Error Correcting Binary Group Codes," Information and Control, 3, 279-290 (1960).

13. Hocquenghem, A., "Codes correcteurs d'erreurs," Chiffres, 2, 147-156 (1959).

14. Bartee, T.C., and Schneider, D.I., "An Electronic Decoder for Bose-Chaudhuri-Hocquenghem Error-Correcting Codes," I.R.E. Transactions, IT-8, 5, 17-24, (1962).

15. Gorenstein, D., Peterson, W.W. and Zieler, N., "Two Error-Correcting Bose-Chaudhuri Codes are Quasi-Perfect," Information and Control, 3, 291-294 (1960).

16. Hammersley, J.M., and Handscomb, D.C., Monte Carlo Methods. New York: Wiley, 1964.

17. Lehmer, D.H., "Mathematical Methods in Large-Scale Computing Units," Proc. Second Symp. on Large-Scale Digital Calculating Mach., 1949.

```
-COOP,0607,HOLT RH,O/49/S/1S/2S/E/45=54,15,10000.BEC PARITY
-FTN,L,E.
      PROGRAM BEC PAR
C VARY BIT ERROR RATE FOR BINARY ERASURE CHANNEL ODD PARITY BIT CODE.
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING 1 PARITY BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=100),          (COR=0),(NON=0),(I=0),(J=0),(ITOP=10000),
     1(KTOP=1     ),(N=0) ,(IA=1),(PB=.1    )
      TYPE INTEGER COR
      PRINT 500
  500 FORMAT(1H1)
      R=0.12125
      CALL RANFSET(R)
   20 DO 12 K=1,KTOP
      DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NB
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-1)6,5,6
    5 COR=COR+1
      GO TO 10
    6 NON=NON+1
   10 CONTINUE
   12 CONTINUE
      ANON=NON
      PF=ANON/(    ITOP*KTOP)
      PRINT 200,PB,NB,COR,NON,PF
  200 FORMAT(//,10X,3HPB=E9.5,4X,3HNB=I7,4X,4HCOR=I7,/,10X,4HNON=I7,4X,
     13HPF=E10.9)
      N=0$NON=0$COR=0
      IF(PB.LE.0.001    )8,9
```

```
    9 PB=PB/10              $   GO TO 20
    8 CONTINUE
      IF(IA.LE.3          )7,15
    7 IA=IA+1   $  PB=0.1
      GO TO 20
   15 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE
```

```
-COOP,0607,HOLT RH,0/49/S/1S/2S/E/45=54,15,10000,BEC PARITY
-FTN,L,E.
      PROGRAM BEC PAR
C  VARYING BLOCK LENGTH FOR BINARY ERASURE CHAN. ODD PARITY CODE
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING 1 PARITY BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=20),(PB=.01 ),(COR=0),(NON=0),(I=0),(J=0),(ITOP=10000),
     1(KTOP=1),(      N=0),(IA=1)
      TYPE INTEGER COR
      R=0.33345
      CALL RANFSET(R)
      PRINT 500
  500 FORMAT(1H1)
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NB
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N.EQ.1)5,6
    5 COR=COR+1
      GO TO 10
    6 NON=NON+1
   10 CONTINUE
      ANON=NON
      PF=ANON/(    ITOP*KTOP)
      PRINT 200,PB,NB,COR,NON,PF
  200 FORMAT(//,10X,3HPB=E9.5,4X,3HNB=I7,4X,4HCOR=I7,/,10X,4HNON=I7,4X,
     13HPF=E10.9)
      N=0$NON=0$COR=0
      IF(NB.GE.100      )15,7
    7 NB=NB+20
      GO TO 20
```

```
   15 CONTINUE
      IF(IA-3)17,18,18
   17 CONTINUE
      IA=IA+1 $ NB=20     $ GO TO 20
   18 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE
```

```
-COOP,0607,HOLT RH,O/49/S/1S/2S/E/45=54,15,10000. HAMMING
-FTN,L,E.
      PROGRAM HAMMING
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH WITH KB PARITY BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=7   ),(PB=0.1),(COR=0),(NON=0),(I=0),(J=0),(ITOP= 5000),
     1(KTOP=1     ),(N=0) ,(IA=1),(KB=3 )
      TYPE INTEGER COR
      R=0.52345
      PRINT 500
  500 FORMAT(1H1)
      CALL RANFSET(R)
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NB
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-1)5,5,6
    5 COR=COR+1
      GO TO 10
    6 NON=NON+1
   10 CONTINUE
      ANON=NON
      PF=ANON/(    ITOP*KTOP)
      ANB=NB
      RD=(ANB-KB   )/ANB
      PRINT 200,PB,NB,COR,NON,RD,PF
  200 FORMAT(//,1X,3HPB=E9.5,3X,3HNB=I5,3X,4HCOR=I5,3X,4HNON=I5,3X,/,1X,
     13HRD=E9.8,3X,3HPF=E10.9)
      N=0$NON=0$COR=0
      IF(NB.GE.63.    )15,7
    7 KB=KB+1    $  NB=2**KB-1
```

```
      GO TO 20
   15 CONTINUE
      IF(IA-3)17,17,18
   17 CONTINUE
      IA=IA+1  $  NB =7 $  KB=3
      GO TO 20
   18 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE,15.
```

```
-COOP,0607,HOLT RH,O/49/S/1S/2S/E/45=54,15,10000. GOLAY
-FTN,L,E.
      PROGRAM GOLAY
C VARY BIT ERROR RATE
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING 12 INFO BITS. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=23 ),(PB=.05),(COR=0),(NON=0),(I=0),(J=0),(ITOP= 5000),
     1(KTOP=2    ),(N=0) ,(IA=1)
      TYPE INTEGER COR
      R=0.52345
      PRINT 500
  500 FORMAT(1H1)
      CALL RANFSET(R)
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NB
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-3) 5,5,6
    5 COR=COR+1
      GO TO 10
    6 NON=NON+1
   10 CONTINUE
      ANON=NON
      PF=ANON/ITOP
      PRINT 200,PB,NB,COR,NON,PF
  200 FORMAT(//,3HPB=E9.5,4X,3HNB=I7,4X,4HCOR=I7,4X,4HNON=I7,4X,
     13HPF=E10.9)
      N=0$NON=0$COR=0
      IF(PB- 0.5 ) 8,8,9
    8 PB=PB + 0.05
      GO TO 20
```

```
    9 IF(IA.GE.3    )15,7
    7 IA=IA+1  $  PB=0.05
      PRINT 500
      GO TO 20
   15 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE,15.
```

```
-COOP,0607,HOLT RH,0/49/S/1S/2S/E/45=54,15,10000. BOSE-CHAUDHURI
-FTN,L,E.
      PROGRAM BOSE
C BOSE CHAUDHURI FOR NB=31 AND T=5 ERRORS.
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING KB INFO. BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=31 ),(PB=0.1),(COR=0),(NON=0),(I=0),(J=0),(ITOP= 1000),
     1(KTOP=1    ),(N=0) ,(IA=1),(KB=11),(MC=5),(MB=20),(T=5  ),
     2(MCMAX= 5),(PBMIN=0.01  ),(TMAX= 5.)
      TYPE INTEGER COR,T
      R=0.52345
      PRINT 500
  500 FORMAT(1H1)
      CALL RANFSET(R)
   30 NB=2**MC-1
      IF(KB-3)32,33,33
   32 T=1 $ N=0 $ NON=0 $   COR=0
      MC=MC+1
      GO TO 30
   33 AKB=KB  $  RD=AKB/NB
      ITOP=100/PB
      NN=NB
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NN
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-T)5,5,6
    5 COR=COR+1
      GO TO 10
    6 NON=NON+1
   10 CONTINUE
```

```
      ANON=NON
      PF=ANON/(    ITOP*KTOP)
      PRINT 200,PB,NB,KB,T,COR,NON,RD,PF
  200 FORMAT(//,1X,3HPB=E9.5,3X,3HNB=I5,3X,3HKB=I5,3X,2HT=I5,/, 1X,
     14HCOR=I5,3X,4HNON=I5,3X,3HRD=E9.5,3X,3HPF=E10.9)
      N=0$NON=0$COR=0
   13 CONTINUE
      IF(IA-3 )14,15,15
   14 IA=IA+1  $  T=5  $  MC=5
      GO TO 30
   15 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE,15.
```

```
-COOP.0607.HOLT RH.0/49/S/1S/2S/E/45=54,15,10000. BOSE-CHAUDHURI
-FTN.L.E.
      PROGRAM BOSE
C BOSE CHAUDHURI FOR NB=31 AND T=7 ERRORS.
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING KB INFO. BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=31 ),(PB=0.1),(COR=0),(NON=0),(I=0),(J=0),(ITOP=10000),
     1(KTOP=1     ),(N=0) ,(IA=1),(KB=6 ),(MC=5),(MB=20),(T=7   ),
     2(MCMAX= 5),(PBMIN=0.01  ),(TMAX= 7.)
      TYPE INTEGER COR,T
      R=0.52345
      PRINT 500
  500 FORMAT(1H1)
      CALL RANFSET(R)
   30 NB=2**MC-1
      IF(KB-3)32,33,33
   32 T=1 $ N=0 $ NON=0 $    COR=0
      MC=MC+1
      GO TO 30
   33 AKB=KB  $  RD=AKB/NB
      NN=NB
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NN
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-T)5,5,6
    5 COR=COR+1
      GO TO 10
    6 NON=NON+1
   10 CONTINUE
      ANON=NON
```

```
      PF=ANON/(    ITOP*KTOP)
      PRINT 200,PB,NB,KB,T,COR,NON,RD,PF
  200 FORMAT(//,1X,3HPB=E9.5,3X,3HNB=I5,3X,3HKB=I5,3X,2HT=I5,/, 1X,
     14HCOR=I5,3X,4HNON=I5,3X,3HRD=E9.5,3X,3HPF=E10.9)
      N=0$NON=0$COR=0
   13 CONTINUE
      IF(IA-3 )14,15,15
   14 IA=IA+1  $  T=7  $  MC=5
      GO TO 30
   15 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE,15.
```

```
-COOP,0607,HOLT RH,0/49/S/1S/2S/E/45=54,15,10000. BOSE-CHAUDHURI
-FTN,L,E.
      PROGRAM BOSE
C PROGRAM FOR BOSE CHAUDHURI CODE W/ NB=7 TO 31. T=ERRORS.
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING KB INFO. BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=7   ),(PB=0.1),(COR=0),(NON=0),(I=0),(J=0),(ITOP= 1000),
     1(KTOP=1     ),(N=0) ,(IA=1),(KB=4 ),(MC=3),(MB=3 ),(T=1   ),
     2(MCMAX= 5),(PBMIN=0.01   ),(TMAX= 5.)
      TYPE INTEGER COR,T
      R=0.52345
      PRINT 500
  500 FORMAT(1H1)
      CALL RANFSET(R)
   30 NB=2**MC-1
   31 MB=MC*T
      KB=NB-MB
      IF(KB-3)32,33,33
   32 T=1 $ N=0 $ NON=0 $    COR=0
      MC=MC+1
      GO TO 30
   33 AKB=KB  $  RD=AKB/NB
      ITOP=100/PB
      NN=NB
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NN
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-T)5,5,6
    5 COR=COR+1
      GO TO 10
```

```
    6 NON=NON+1
   10 CONTINUE
      ANON=NON
      PF=ANON/(    ITOP*KTOP)
      PRINT 200,PB,NB,KB,T,COR,NON,RD,PF
  200 FORMAT(//,1X,3HPB=E9.5,3X,3HNB=I5,3X,3HKB=I5,3X,2HT=I5,/,
     14HCOR=I5,3X,4HNON=I5,3X,3HRD=E9.5,3X,3HPF=E10.9)
      N=0$NON=0$COR=0
      AT=T
      IF(AT-  TMAX) 12,13,13
   12 T=T 1
      GO TO 31
   13 CONTINUE
      IF(IA-3 )14,15,15
   14 IA=IA+1  $  T=1  $  MC=3
      PRINT 500
      GO TO 30
   15 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE,15.
```

```
-COOP,0607,HOLT RH,0/49/S/1S/2S/E/45=54,15,10000. BOSE-CHAUDHURI
-FTN,L,E.
      PROGRAM BOSE
C BOSE-CHAUDHURI FOR NB=63 AND T=U TO 7. WHERE T = ERRORA.
C PB=BIT ERROR PROB. QB=1-PB=BIT CORRECT PROB. PF=FINAL ERROR RATE.
C NB=BLOCK LENGTH USING KB INFO. BIT. COR=COR. BLOCKS. NON=NON-COR. BLOCKS
      DATA(NB=63 ),(PB=0.1),(COR=0),(NON=0),(I=0),(J=0),(ITOP= 5000),
     1(KTOP=1     ),(N=0) ,(IA=1),(KB=57),(MC=6),(MB=6 ),(T=1   ),
     2(MCMAX= 6),(PBMIN=0.01  ),(TMAX= 7.)
      TYPE INTEGER COR,T
      R=0.12345
      PRINT 500
  500 FORMAT(1H1)
      CALL RANFSET(R)
   30 NB=2**MC-1
   31 MB=MC*T
      KB=NB-MB
      IF(KB-3)32,33,33
   32 T=1 $ N=0 $ NON=0 $    COR=0
      MC=MC+1
      GO TO 30
   33 AKB=KB  $  RD=AKB/NB
      ITOP=100/PB
      NN=NB
   20 DO 10 I=1,ITOP
      N=0
      DO 11 J=1,NN
      X=RANF(-1)
      IF(X.LT.PB)3,11
    3 N=N 1
   11 CONTINUE
      IF(N.EQ.0)10,4
    4 CONTINUE
      IF(N-T)5,5,6
    5 COR=COR+1
      GO TO 10
```

```
    6 NON=NON+1
   10 CONTINUE
      ANON=NON
      PF=ANON/(    ITOP*KTOP)
      PRINT 200,PB,NB,KB,T,COR,NON,RD,PF
  200 FORMAT(//,1X,3HPB=E9.5,3X,3HNB=I5,3X,3HKB=I5,3X,2HT=I5,/, 1X,
     14HCOR=I5,3X,4HNON=I5,3X,3HRD=E9.5,3X,3HPF=E10.9)
      N=0$NON=0$COR=0
      AT=T
      IF(AT-  TMAX) 12,13,13
   12 T=T 1
      GO TO 31
   13 CONTINUE
      IF(IA-3 )14,15,15
   14 IA=IA+1  $  T=1  $  MC=6
      PRINT 500
      GO TO 30
   15 CONTINUE
      PRINT 500
      END
      END
         FINIS
-EXECUTE,15.
```

INITIAL DISTRIBUTION LIST

		No Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library Naval Postgraduate School Monterey, California 93940	2
3.	Naval Ships System Command Department of the Navy Washington, D.C. 20360	1
4.	Prof. C. F. Klamm, Jr. (Thesis Advisor) Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
5.	LT Richard Harold Holt, USN Box 2526 Naval Postgraduate School Monterey, California 93940	2

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
Monte Carlo Determination of Bounds on Error Correcting Codes

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Thesis, M.S., June 1967

5. AUTHOR(S) *(Last name, first name, initial)*
HOLT, Richard Harold

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1967	72	17

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES
[redacted]

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Ship Systems Command

13. ABSTRACT

Analytical bounds on the capabilities of error correcting codes have been found for most known codes. There are only a limited number of coding theorists available for the solution of such problems; however, new coding techniques are constantly being proposed to meet new communication system problems. This paper develops and describes a Monte Carlo method for estimating bounds experimentally. A bound on the block error rate is developed for use in evaluating optimum codes. The random sampling technique evolved is used to evaluate the bounds on four representative error correcting codes. The close agreement between the theoretical and experimental results establishes confidence in the method's use to determine bounds and capabilities. The technique and problems described can also be used to simulate error correction in system coding studies.

DD FORM 1 JAN 64 1473

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Error Correction						
Error Correction Codes						
Monte Carlo Method						
Bounds on Error Correction Codes						